Secrets Inside the Linux Kernel Revealed

<< prev page 1 2 3 4 next >>

Virtual Memory

Earlier, I said that the virtual memory subsystem in the kernel w memory protection (preventing processes from interfering with ‹ memory) and paging (allowing memory pages to be temporarily disk). This is easily one of the most complex aspects of the Linu and it bears taking a closer look at. Understanding how virtual n Linux is the key to understanding many other features, such as tl

This is a topic which has consumed a fair number of pages in bo advanced operating systems textbooks (not to mention a slew of and articles in magazines such as *Glamour*), so I can't expect to bulk of virtual memory management in this article. Hopefully, tl you a clear picture of what the various pieces are so you can loo more detail elsewhere. Different operating systems implement v management in very different ways, so my treatment here will b Linux. It's also helpful to focus on a particular hardware architec Linux relies heavily (as do other operating systems) on the mem support provided by the hardware. In this case, I'll focus on the I discussion is similar for other systems.

The very meaning of the term virtual memory is that an applicat the illusion that it has access to a much larger amount of memor present on the system. Ideally, we'd like the application to belie entire range of memory addresses allowed by the CPU -- which 2^{32} bytes, or 4 gigabytes of virtual memory. (This is because ev "virtual address", is stored in 32 bits and each bit only has 2 pos 2^{32}, then, is the range of virtual memory which can be addresse pointer. We can write this range, in hexadecimal, as 0x 0000000 Very few systems, however, have 4 gigabytes of physical memo addition, multiple applications may be running on the system at each application to be able to access the entire 4-gigabyte "virtu without interfering with one another. How are we going to acco

Luckily, the Intel x86 architecture (as well as most other moder architectures) includes hardware support for implementing virtu

Reviews
Shutdown
How To
LAMP Post
Power Tools
Guru Guidance
On The Desktop
Best Defense
Developer's Den
Java Matters
Extreme Linux
Compile Time
Perl of Wisdom
Gearheads Only
Project of the Month
Who's Who
Downloads

form of a "memory management unit", or MMU. The MMU is r
translating virtual memory accesses made by user programs into
accesses of the actual RAM in the machine. Of course, the MMI
the operating system to do this -- which is where the Linux kern
subsystem comes into play. Before we get into all of that, howe
what happens when a user program reads or writes an address in

Translating virtual addresses

Let's say that a user process (such as Emacs) wants to read or wı
virtual address 0x 48b32f00 (this address is meaningless to a hu
course, but to Emacs it might represent the current position of th
window). In order to translate this memory reference into an adc
memory, the MMU uses a special set of structures, called the pa
specify for each page (4 kilobytes) of virtual memory what phys
address (if any) should be used. We can think of a particular pag
containing the following information:

```
virtual page address -> physical page address (+ oth
```

So, there might be an entry in the page tables which looks like:

```
0x48b32000 -> 0x0028a000 (+ other information)
```

Note that the MMU deals with memory in page-sized chunks, w
things) reduces the size of the page tables themselves. Since a pa
the last 3 digits (in hexadecimal) of the virtual and physical page
always '000'. The last three digits of an address (such as 0x48b3:
offset into the page which the user is accessing; in this case that
The MMU adds the page offset to the physical page address fou
tables to produce a complete physical memory address -- *voila!*

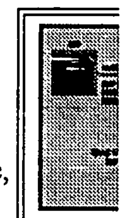
Certainly not every virtual address in the entire 4 GB range corr
physical memory address -- unless one has 4 GB of memory ins
these page table entries might contain a physical page address w
"invalid", meaning that there is no physical page corresponding
When an invalid entry is read by the MMU, a page fault occurs.
about this case later.

Page tables and the TLB

Note that the page tables are actually stored in memory themsel
interesting issue, as it means that the MMU might have to consu
order to look up something else in the page tables. This issue is
fact that there is not just a single page table -- rather, there are th
tables which must be traversed by the MMU in order to translate
address into a physical address. This is done for space reasons; c
single page table entry actually consumes four bytes. If we have

entry per page of virtual address space, that's ((4 Gb/4 Kb) * 4 b megabytes of memory, just to hold the page tables for a single u multiple levels of page tables actually allows the hardware to sw page tables out to disk when they're not being used -- we warnec going to get hairy! For now, though, don't worry about it -- it sui that there is a single (in-memory) page table being consulted by every memory reference.

If the MMU is performing multiple memory operations just to tr address for another memory operation, clearly this is going to be performance. To remedy this problem, the MMU hardware inclu for virtual-to-physical memory translations, called the translatio or TLB. The TLB can be thought of as containing a very small p tables in very fast RAM to speed up MMU address translations.



How th access from a addres

Figure 2 shows what happens during a single memory access by a user application. On the upper left of the figure we have the CPU, from which a user process wishes to read the virtual address 0x48b32f00. First, the TLB is consulted, and if a translation is found there, the physical address is used to access memory directly. (We've drawn the TLB as being separate from the CPU itself, but it's actually part of the processor in most cases.) If the TLB does not contain a mapping for the given virtual address, the MMU must then perform the arduous task of looking up the address in the page tables. However, the result of this lookup will be saved in the Tl increasing performance if another address on the same page is a

So far I've talked about the Intel x86's MMU hardware, not abou Clearly, the kernel isn't involved every time a virtual address is t would be far too slow. So, what does the kernel have to do with management?

The first job of the kernel is to set up and maintain the page tabl will traverse when translating addresses. This requires the kerne how physical memory is laid out, to allocate portions of it to var processes, and to create page table entries allowing virtual addre process to eventually map onto physical RAM.

<< prev page 1 2 **3** 4 next >>